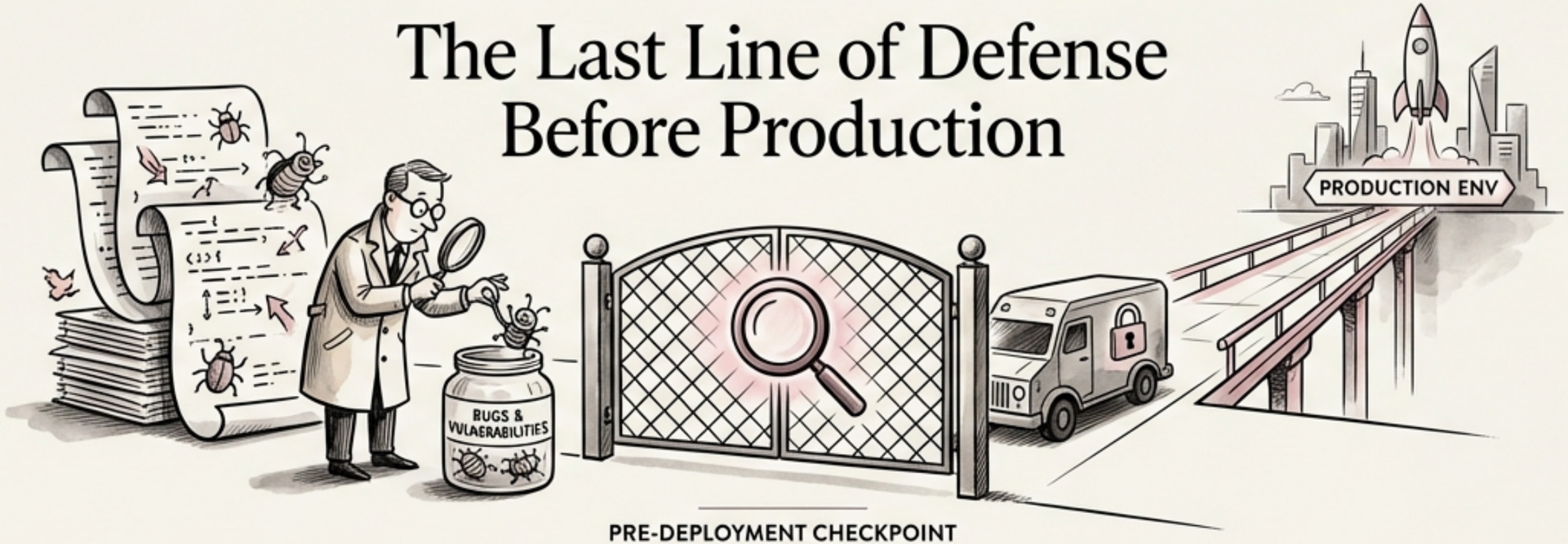
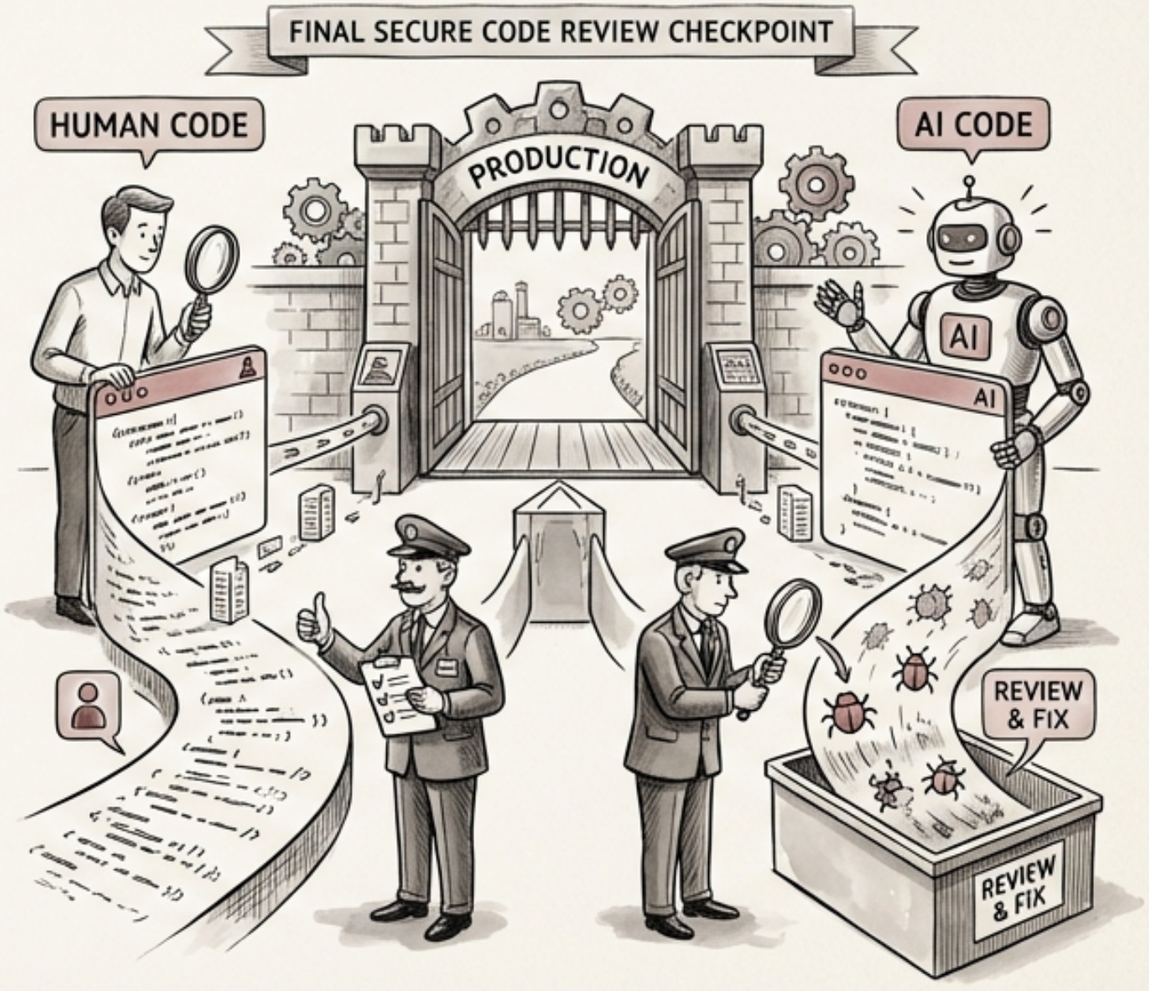
SECURE CODE REVIEW: THE LAST LINE OF DEFENSE BEFORE PRODUCTION

The Last Line of Defense Before Production



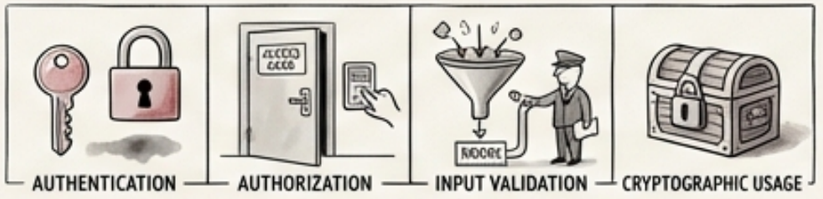
Secure Code Review: The Last Line of Defense Before Production

- Code review serves as the final human checkpoint before code is deployed to production environments.
- In AI-augmented teams, AI-generated code requires scrutiny equal to or greater than human-written code.
- AI-generated code can appear confidently correct but may harbor subtle vulnerabilities.
- A robust secure code review process is critical for mitigating potential risks introduced by both human and AI-written code.
- This presentation outlines a 10-point checklist, AI-assisted tools, and best practices for secure code review.



10-POINT SECURE CODE REVIEW CHECKLIST: A COMPREHENSIVE APPROACH

- The 10-point checklist provides a structured approach to identifying potential security vulnerabilities.



- It covers key areas such as authentication, authorization, input validation, and cryptographic usage.

- The checklist aims to ensure adherence to secure coding practices and architectural guidelines.

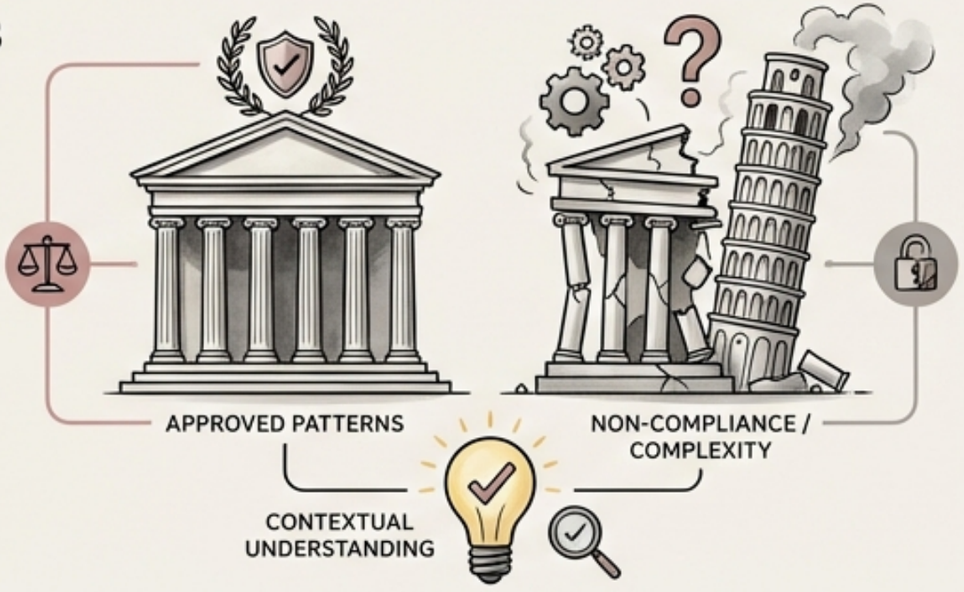
- Each point focuses on a specific aspect of code security and requires thorough examination.

- This checklist serves as a foundation for comprehensive secure code reviews.

Change Context and Architecture Alignment: Understanding the Big Picture

PRESENTATION CONTENT SLIDE

- **Understand the change context:** Determine the problem the code solves and the associated threat model.
  - Consider the potential attack vectors and how the code might be exploited.
- **Verify architecture alignment:** Ensure the code follows approved design patterns and security principles.
  - Non-compliance with architecture can introduce vulnerabilities and increase complexity.
- **Contextual understanding** provides a deeper appreciation of the change's impact and risk.



Authentication and Authorization: Securing Access Controls

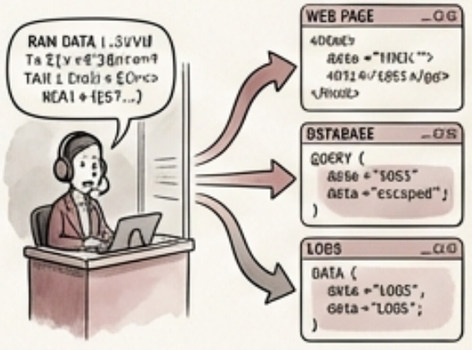
- **Review authentication and authorization logic:** Scrutinize access controls to ensure they are correct and complete.
- Verify that only authorized users have access to specific resources and functionalities.
- Common flaws include insecure password storage, weak authentication mechanisms, and authorization bypasses.
- Consider role-based access control (RBAC) and ensure proper implementation.
- Pay special attention to edge cases and potential privilege escalation vulnerabilities.

Input Validation and Output Encoding: Preventing Injection Attacks

- **Verify input validation and output encoding:** Ensure every input is validated and every output is encoded for its context.

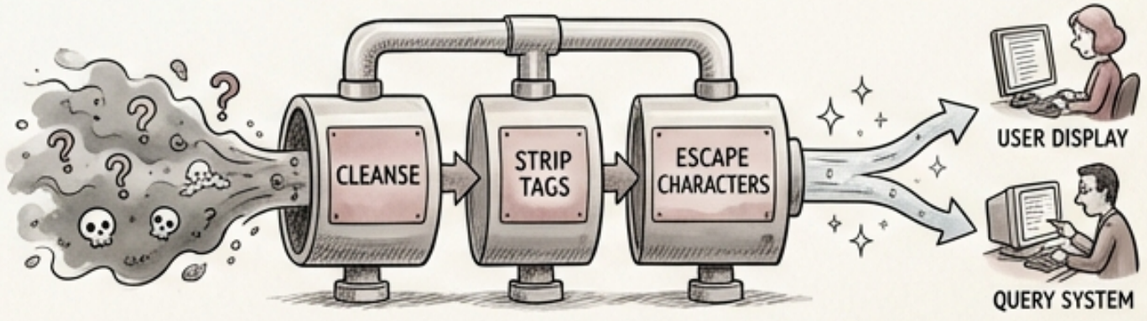
- This prevents common vulnerabilities like SQL injection, cross-site scripting (XSS), and command injection.

- Use **allow-lists** instead of **block-lists** for input validation whenever possible.



- **Encode output** based on the **context**, such as HTML encoding for web pages.

- **Sanitize data** before it's used in queries or displayed to users.



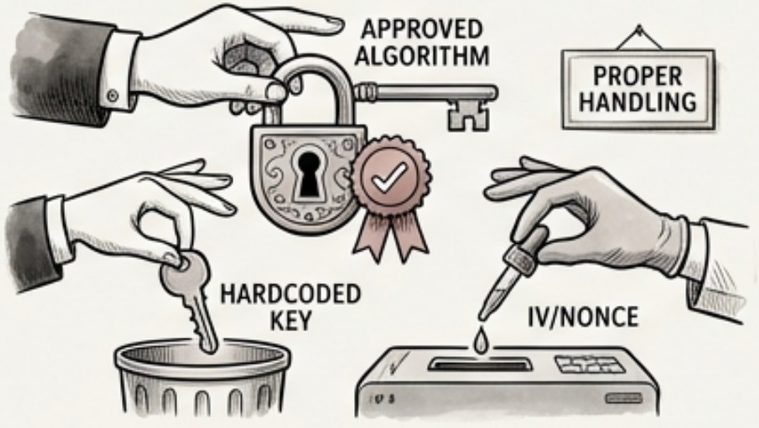
Error Handling and Cryptographic Usage: Minimizing Information Disclosure and Ensuring Data Security

- **Check error handling:** Ensure no stack traces are leaked, no information is disclosed, and graceful degradation is achieved.
- Avoid displaying sensitive information in error messages.

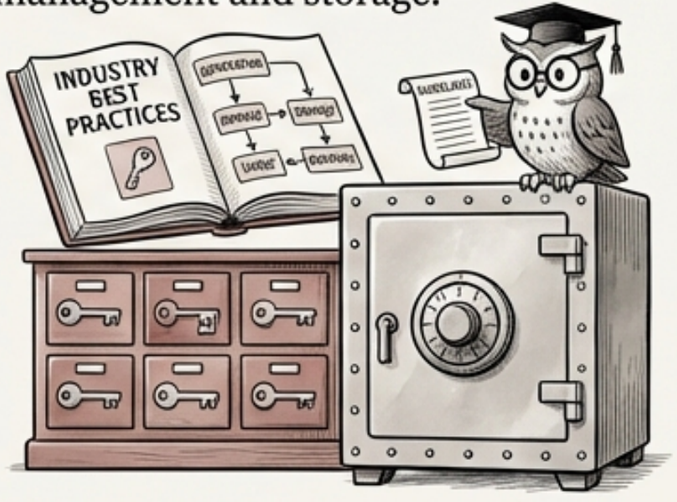
- Avoid displaying sensitive information in error messages.

- Log errors securely and monitor for unusual patterns.

- **Review cryptographic usage:** Use approved algorithms only, avoid hardcoded keys, and handle IVs/nonces properly.

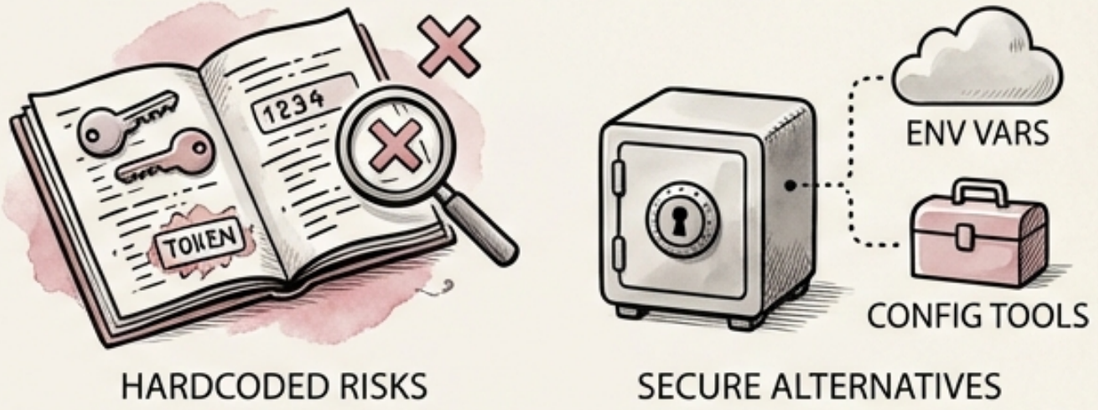


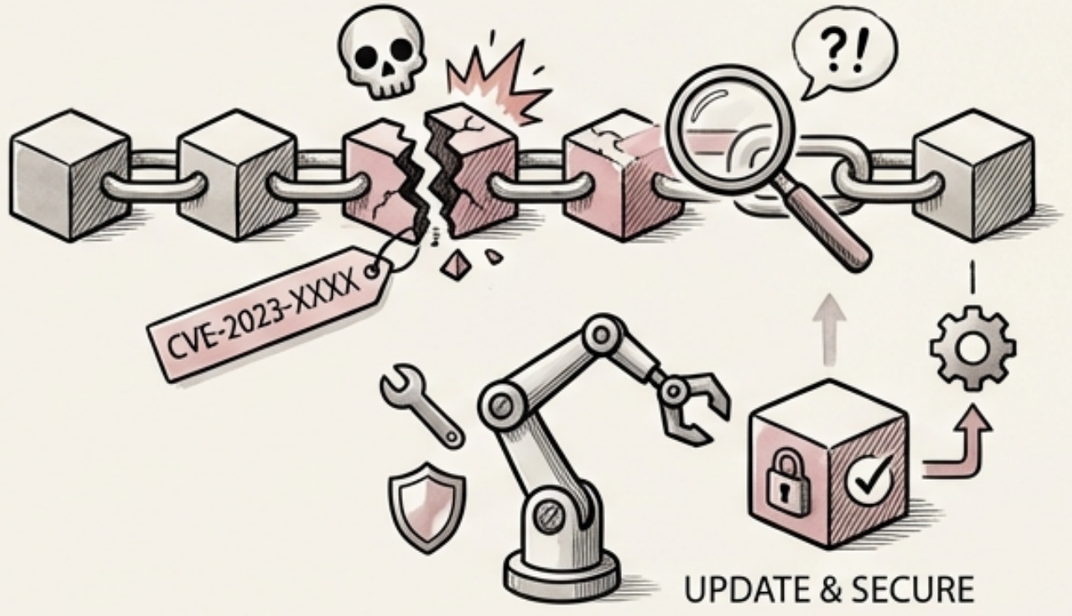
- Follow industry best practices for key management and storage.



HARDCODED SECRETS AND DEPENDENCY SAFETY: ELIMINATING EASY TARGETS

- **Check for hardcoded secrets:** Ensure no API keys, passwords, tokens, or connection strings are present in the code.
- Use environment variables or secure configuration management tools instead.
- **Verify dependency safety:** Ensure all dependencies exist, are pinned, and have no known CVEs.
- Use dependency scanning tools to identify vulnerable dependencies.
- Keep dependencies up-to-date with the latest security patches.

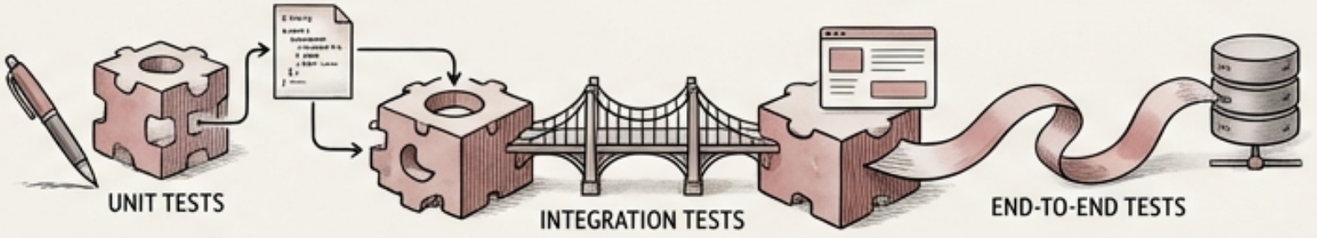




Test Coverage and Business Logic Flaws: Ensuring Functionality and Security

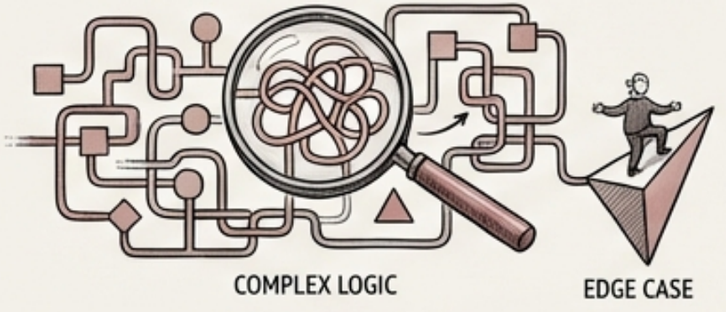
- **Assess test coverage:** Ensure security-relevant paths are tested, edge cases are covered, and negative tests are present.

- Write unit tests, integration tests, and end-to-end tests to cover different aspects of the code.

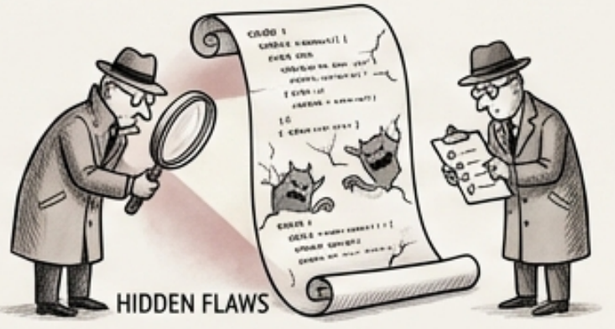


- **Review for business logic flaws:** Identify authorization bypasses, race conditions, and financial calculation errors.

- Pay close attention to complex logic and edge cases.

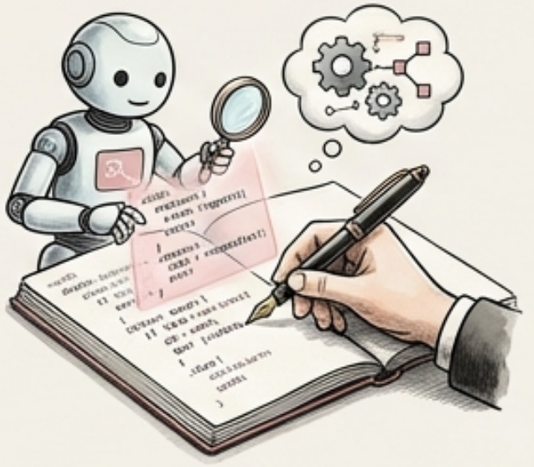


- Perform thorough code review and testing to uncover hidden flaws.



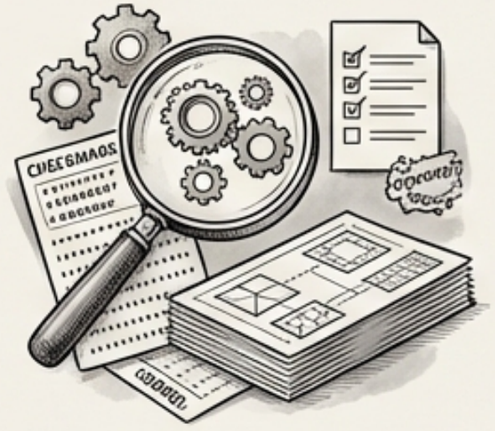
AI-Assisted Code Review Tools: Augmenting Human Expertise

- **GitHub Copilot Code Review:** Offers inline suggestions during PR review and pattern-based detection.
- **CodeRabbit:** Provides automated PR review with a security focus and architectural analysis.
- **Semgrep:** A rule-based SAST tool with custom rule support for organizational patterns.
- **SonarQube:** Provides continuous code quality and security analysis.
- **Snyk Code:** Offers real-time vulnerability detection during development.



AI SAST vs Traditional SAST: A Hybrid Approach

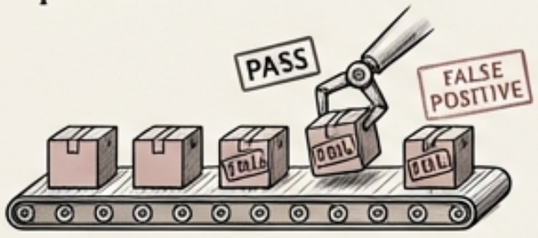
Integrating Deterministic Rules with Probabilistic Learning for Comprehensive Security



(Semgrep, CodeQL, Checkmarx)

TRADITIONAL SAST

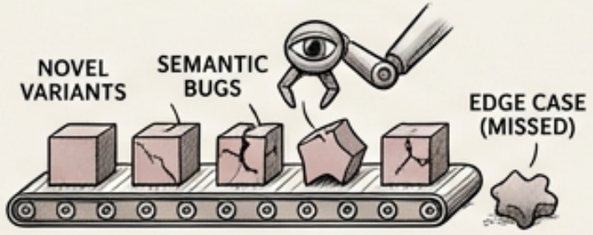
- Rule-based, deterministic, well-understood coverage, low false negative rate but high false positive rate, catches all known patterns.



(Snyk Code, DeepCode, Claude)

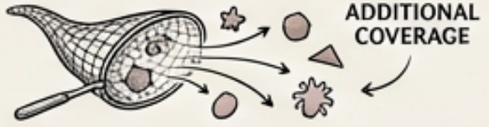
AI-POWERED SAST

- Pattern-learning, probabilistic, catches novel variants and semantic bugs, lower false positive rate but may miss edge cases, finds issues traditional tools cannot.



Traditional SAST acts as a baseline guarantee, catching known vulnerabilities reliably.

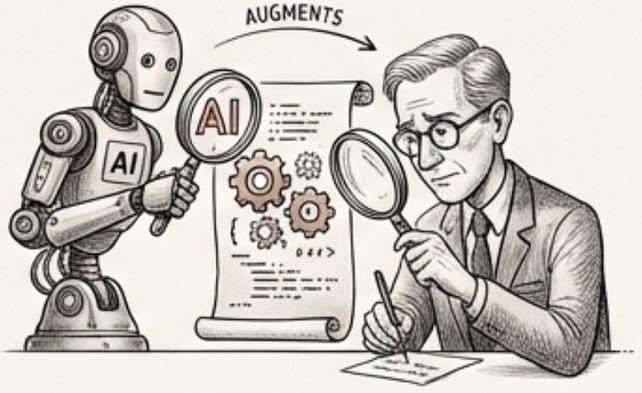
AI SAST offers additional coverage by detecting novel variants and semantic bugs.

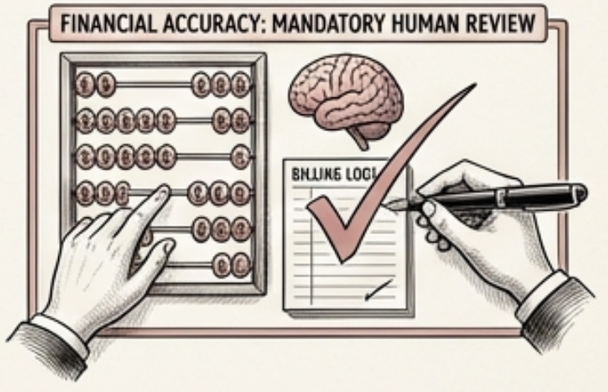


Best practice: run both in the pipeline for comprehensive coverage.

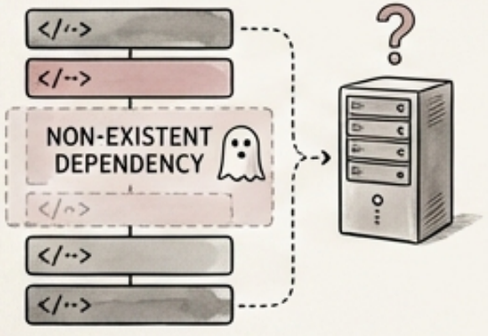
Human-in-the-Loop: Non-Negotiable Requirements for Critical Code

- AI review tools augment human reviewers but never replace them for critical code sections.
- Mandatory human review is required for all authentication and authorization changes.
- Mandatory human review is required for cryptographic implementations and key management.
- Mandatory human review is required for data handling and privacy-sensitive code.
- Mandatory human review is required for financial calculations and billing logic.

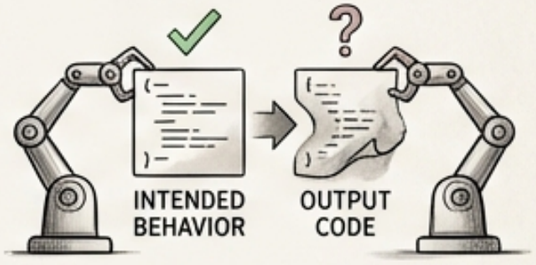




Specific Review Checklist for AI-Generated Code: Going Beyond the Basics



- Beyond the standard 10-point checklist, AI-generated code requires additional scrutiny.
- Was the prompt appropriate for the security context and behavior?
- Does the output actually match the intended behavior or just look like it does?
- Are there hallucinated dependencies that don't exist?
- Are there subtle logic errors masked by confident, well-formatted code style?



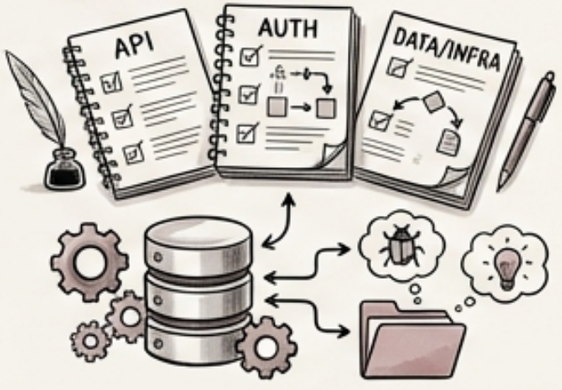
Code Review Metrics and Tracking: Measuring Effectiveness and Identifying Trends

- **Review coverage:** Percentage of changes that receive security review.
- **Mean time to review:** How long from PR creation to security review completion.
- **Defect detection rate:** Vulnerabilities found per 1000 lines reviewed.
- **False positive rate:** Percentage of flagged issues that are not actual vulnerabilities.
- **Review depth:** Average time spent per line of code reviewed.

Building a Security Review Culture: Fostering Collaboration and Responsibility

- **Security champions:** Designated developers on each team trained in secure code review.
- **Review templates:** Standardized checklists for different change types

- **Knowledge sharing:** Weekly security review highlights, common findings database.
- **Blameless culture:** Reviews find vioners.
- AI-generated code is (A\PI, auth, data, infra).

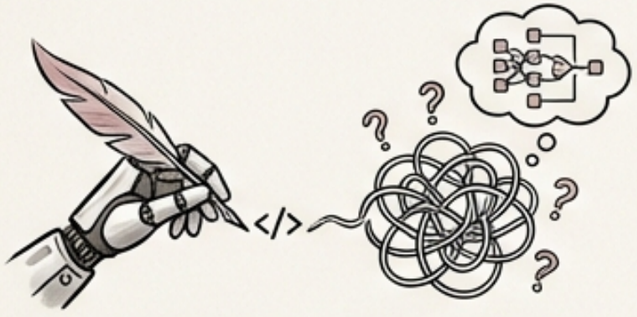


- **Knowledge sharing:** Weekly security review highlights, common findings database.
- **Blameless culture:** Reviews find bugs, not not blame developers.

- AI-generated code is not exempt — the developer who accepts AI suggestions owns the security of that code.

Secure Code Review: A Continuous Journey, Not a Destination

- Secure code review is a critical component of a robust software development lifecycle.



- AI-generated code introduces new challenges that require tailored review processes.

- A combination of human expertise and AI-assisted tools provides the most effective approach.
- Tracking code review metrics helps to improve effectiveness and identify trends.

- Human-in-the-loop is non-negotiable for critical code sections.

THANK YOU

- Questions?